Local Feature Weighting in Nearest Prototype Classification

Fernando Fernández and Pedro Isasi

***Abstract*—The distance metric is the corner stone of nearest neighbor (NN)-based methods, and therefore, of nearest prototype (NP) algorithms. That is because they classify depending on the similarity of the data. When the data is characterized by a set of features which may contribute to the classification task in different levels, feature weighting or selection is required, sometimes in a local sense. However, local weighting is typically restricted to NN approaches. In this paper, we introduce local feature weighting (LFW) in NP classification. LFW provides each prototype its own weight vector, opposite to typical global weighting methods found in the NP literature, where all the prototypes share the same one. Providing each prototype its own weight vector has a novel effect in the borders of the Voronoi regions generated: They become nonlinear. We have integrated LFW with a previously developed evolutionary nearest prototype classifier (ENPC). The experiments performed both in artificial and real data sets demonstrate that the resulting algorithm that we call LFW in nearest prototype classification (LFW-NPC) avoids overfitting on training data in domains where the features may have different contribution to the classification task in different areas of the feature space. This generalization capability is also reflected in automatically obtaining an accurate and reduced set of prototypes.**

***Index Terms*—Evolutionary learning, local feature weighting (LFW), nearest prototype (NP) classification, weighted Euclidean distance.**

I. Introduction

NEAREST PROTOTYPE (NP) classifiers [1] allow to define the class of a new example, usually called query, on the basis of a set of previously classified prototypes. The way of computing this set of prototypes is based on selecting them from an original set of labelled samples, or by replacing the original set by a different and reduced one [2]. Learning vector quantization (LVQ) algorithms [3] are a family of algorithms focused on NP classification. They are based on the dynamical computation of a set of prototypes in order to minimize the classification error. Several approaches are based on this model [4].

It has been proved that NP classifiers have an asymptotic error rate that is at most twice the Bayes error rate, no matter the distance used in the classification [5]. However, this advantage decreases when the dimensionality of the input data increases [6]. Unfortunately, as the dimensionality of the input becomes higher, the choice of the distance metric becomes more important in determining the outcome of the NP classification. A commonly used distance metric is the Euclidean distance. It is based on the assumption that the input space is isotropic or homogeneous. This assumption is not true in many practical domains. Usually, the domains contain a high dimensionality spuriously, there are irrelevant features and those are not homogeneous, and many times the normalization of the space in a global way does not produce any advantage.

To reduce dimensionality, two approaches can be followed. On the one hand, feature selection methods try to identify the important features for the classification process, eliminating the rest of the features [7]. This can be done by projecting the input space into a new lower dimensional one, improving the classification accuracy [8]; or by finding out relationships, linear or nonlinear, among the attributes, that allow a better classification after applying the found transformation [9]. On the other hand, weighting methods try to weight the importance that the distance metric gives to each feature, so different features can receive different treatments [10].

Weighting approaches are typically followed to introduce both feature selection or weighting, using a weighting factor that can be constant, or a function of the example which is being classified. If the weighting factor is constant, it is called global weighting, given that the whole domain will receive the same weighting vector. If the weighting factor is not constant, it is called local weighting, given that the weighting factor applied is a function that typically depends on the example, and hence, on the area of the domain where it is located. An extensive documentation on feature weighting methods for nearest neighbor (NN) classification can be found in [10].

In this paper, we present a method for local weighting in NP classification. The way to do this is to provide the prototypes their own continuous weight vector. Thus, the algorithm follows a prototype-specific weighting approach, instead of a typical instance-specific one [11]. This approach allows for the borders among Voronoi regions to be nonlinear, which provides better generalization capabilities in domains where the relevance of the features may be different in different areas of the space. To do that, we include a new method of computing the weights for each prototype, in such a way that the values of all the components of the distortion vector of a prototype are the same. The distortion vector is defined as the average difference contributed by each training instance belonging to the region with respect to the prototype, as it is explained in Section IV.

This prototype-specific weighting approach could be used in many classification algorithms. Here, the method is implemented as an extension of the evolutionary nearest prototype

Manuscript received April 17, 2006; revised November 10, 2006 and January 31, 2007; accepted February 7, 2007. The work of F. Fernandez was supported in part by the Grant from the Spanish Ministry of Education and Fulbright, the Spanish MEC project TIN2005-08945-C06-05, and the regional CAM-UC3M project CCG06-UC3M/TIC-0831. The work of P. Isasi was supported in part by the MCyT project OpLink TIN2005-08818-C04-02.

The authors are with the Departamento de Informatica, Universidad Carlos III de Madrid, 28911 Leganes, Madrid, Spain (e-mail: ffernand@inf.uc3m.es; isasi@ia.uc3m.es).

Digital Object Identifier 10.1109/TNN.2007.902955

classification (ENPC) algorithm [12], a nearest prototype (NP) approach that follows an evolutionary process to compute a correct number of prototypes. We have called this extension local feature weighting in nearest prototype classification (LFW-NPC). In this new approach, weighting vectors are computed at the same time as prototype locations, given that both are considered as characteristics of the prototypes, and both can be modified in the evolutionary process.

Section II summarizes the related work. Section III defines our method for local feature weighting (LFW) in NP classifiers. Section IV shows how LFW is integrated in the previously developed ENPC algorithm. Section V describes the experiments performed, and Section VI summarizes the main conclusions.

II. Related Work

Reducing dimensionality can be done by projecting the input space into a new low-dimensional "effective" subspace, where the relationships between variables allows a better classification performance. Torkkola [9] proposed using nonparametric estimation of some relationship between input space and output space, so called mutual information, to find an "effective" subspace represented by a matrix that maximizes that relationship. In a similar way, Bach and Jordan [13] show how to use reproducing kernel Hilbert spaces to characterize marginal independence between pairs of variables, and thereby, design an objective function for independent component analysis (ICA). This work has been extended in [8] to achieve conditional independence to generate projection functions to reduce dimensionality. Estébanez *et al.* [14] automatically generate those projection functions by genetic programming, for some specific domains. Some other approaches follow the idea of optimizing the metric of the nearest neighborhood rule. In [15], the authors propose to search this measure in a local way. First, the set of prototypes are extracted in the learning phase, and then, for such a distribution of prototypes, a metric is searched trying to optimize the accuracy.

In the following, we will focus on approaches similar to our proposal, i.e., methods for reducing the dimensionality by using weighting factors in the NN rule. Such methods are typically based on the weighted euclidean distance, defined in[1]

$$d_{\vec{w}}(\vec{x},\vec{y}) = \sqrt{\sum_{i=1}^{K} w_i (x_i - y_i)^2} \tag{1}$$

where $\vec{w} = (w_1, \ldots, w_K)$ is the weighting vector, in the K-dimensional space. For instance, Peng *et al.* [17] introduce a locally adaptive neighborhood morphing classification method to try to minimize the bias due to the curse of dimensionality. The authors use local support vector machine (SVM) learning to estimate an effective metric for producing neighborhoods that are elongated along less discriminant feature dimensions and constricted along most discriminant ones. As a result, the class conditional probabilities can be expected to be approximately constant in the modified neighborhoods, which is the assumption needed for k-NN techniques to work optimally. In this method, the authors uses the weighted Euclidean distance defined in (1), where w_i does not evolve, but it is adapted from the query (so w_i depends on the query x) following:

$$w_i(x) = \frac{e^{cR_i(x)}}{\sum_{l=1}^{n} e^{cR_l(x)}} \tag{2}$$

where c is a constant and R_i is computed by following an SVM scheme. It captures the relevance information along each individual dimension. In this way, the neighborhood changes while decreasing the distance between the query and the decision boundary, making it more elliptical. This method has three main parameters: K or the number of NNs in the final NN rule, K_L or the number of NNs in the final nearest neighborhood for SVM computation, and c or the positive factor for the exponential weighting scheme defined in the (2).

Hastie and Tibshirani [18] iteratively change the weights of the attributes for each query. They select the neighborhood of a query and apply local discriminant analysis to shrink the distance, parallel to the boundary between decision classes. This new metric is then used to select the k-NNs as usual.

Domeniconi *et al.* [19]–[21] use a similar idea, but they use SVM instead of local discriminant analysis to shrink the distance. In [22], they propose an adaptive k-NN classification method, also to minimize the estimation bias in high dimensions. They estimate a flexible metric for computing neighborhoods based on chi-squared distance analysis. This technique is capable of producing a local neighborhood in which the posterior probabilities are approximately constant and that is highly adaptive to query locations. They utilize the same weighted distance computation as in (1) and [18]. The weights are assigned for each query in an iterative process: The weights are initialized to one for each query, and then modified in an iterative process that ends after some few iterations. In a more recent work [23], SVM is used to determine the most discriminant direction in a neighborhood around a query, providing also a LFW scheme.

The aforementioned methods are instances of lazy learning methods, and need to execute the whole training procedure for each query. That makes k-NN algorithms impractical when fast answers are required over large databases, given that answer time should be very long. This problem is typically mitigated using different organization methods as trees [24]. However, when the distance metric depends on the query, it is not clear how to perform such an organization.

Another way to minimize the answer time of a query is by using the NP classification, which represents the whole data set with a reduced set of prototypes. Feature weighting has also been applied to the NP approaches, and some methods have been proposed to adapt metrics during training. For instance, distinction sensitive learning vector quantization (DSLVQ) [25] automatically determines weighting factors to the input dimension of the training data. These weights are adapted according to some heuristics.

Hammer and Villmann [26] try to automatically scale the input dimensions and to adapt the Euclidean metric to the specific training problem. The idea is to introduce weighted factors that could be adapted automatically to minimize the classification errors. This scheme is similar to the LVQ2 where the NP is attracted to the query if it belongs to the correct class,

[1]An extensive discussion on different distance metrics for local weighting can be found in [16].

and it is repelled otherwise. LVQ2.1 works similarly, and the nearest correct prototype is attracted while the nearest wrong prototype is repelled. Different authors use a similar scheme, where attraction and repulsions are implemented by sigmoidal functions. This leads to the generalized learning vector quantization (GLVQ) method [27]. However, the computation of distances for selecting the NPs are still based on Euclidean distance. A modification of this method, called relevance learning vector quantization (RLVQ), has been proposed in [28], where the computation of the distance is also weighted [following (1)] and the weights are updated following:

$$w_i = \begin{cases} w_i - \mu\left(x_i^j - p_i^k\right)^2, & \text{if } x_i^j \text{ and } p_i^k \text{ same class} \\ w_i + \mu\left(x_i^j - p_i^k\right)^2, & \text{otherwise.} \end{cases} \quad (3)$$

This same scheme is used in generalized relevance learning vector quantization (GRLVQ), where the updating of the weights changes, using the same sigmoidal function as for the updating of the prototypes.

In all those last cases, authors are not using a lazy perspective. However, they follow a global weighting approach. In this paper, by the opposite, we describe an approach for LFW in a nonlazy way, by following the NP approach. The idea is to provide each prototype with its own weight vector, so each prototype is given a different distance metric. Thus, the distance metric depends on the prototype and not the query, as the previously mentioned methods for LFW in k-NN do.

There are some previous approaches for local metric adaptation in NP classification. For instance, soft nearest prototype classification with relevance learning (SNPC-R) adapts its metric providing the distance measure with a weighting vector parameter α. That work describes the idea of using an individual parameter α^r for each prototype w_r (like LFW-NPC does) or a classwise metric shared within prototypes with the same class label. The second approach was implemented resulting in the LSNPC-R algorithm [29].

There are other clustering methods that use adaptive metrics for each prototype. For instance, the fuzzy clustering algorithms Gustafson–Kessel [30] and Gath–Geva [31] compute the distance of any data point x from the cluster prototype L^i as defined in

$$d(x, L^i) = (x - L^i)M_i(x - L_i) \quad (4)$$

with M_i symmetric and positive definite. The matrices M_is allow variations in the shape of the clusters, like our approach does, although our approach is for supervised learning.

Providing each prototype with its own weight vector generates nonlinear Voronoi regions among the different prototypes. These nonlinear regions are computed from a set of training instances, in a nonlazy way, and are afterward used to classify future queries. The definition of nonlinear Voronoi regions provides the classifier with a more accurate definition of the borders between categories, and improves the performance of predicting the class to whom the queries belong. This new local weighted NP (nonlazy) perspective, and the way of computing the weights, are explained in Section III.

III. Local Feature Weighting

LFW-NPC requires a set of prototypes $C = \{r_1, \ldots, r_N\}$. Each of these prototypes is composed of its location, class, and weight vector, so $r_i = \langle \vec{p_i}, s_i, \vec{w_i} \rangle$, where $\vec{p_i} \in \mathcal{R}^K$, $s_i \in S$ (the set of all the possible classes), and $\vec{w_i} = \{w_{i1}, \ldots, w_{iK}\}$, with $w_{ij} \in [0, \infty)$. Next, we include a formal description of the distortion measure used and the NP rule.

A. Distortion Measure and the NP Rule

Weighted Euclidean distance, defined in (5), is typically applied to global weighting in NNs methods

$$d_{\vec{w}}(\vec{x}, \vec{y}) = \sqrt{\sum_{i=0}^{i<K} (\vec{x}[i] - \vec{y}[i])^2 * \vec{w}[i]} \quad (5)$$

where K is the dimension of the data, $\vec{x}$ and $\vec{y}$ are the elements for which we are computing the distance, and $\vec{w}$ is the weighting vector. This vector is the same for the whole feature space. However, to introduce local weighting, we are interested in using a different weighting vector for measuring the similarity of any instance to each prototype. This new similarity function $d_{\vec{w}(\vec{p})}$ is defined in

$$d_{\vec{w}(\vec{p})}(\vec{x}, \vec{p}) = \sqrt{\sum_{i=0}^{i<K} (\vec{x}[i] - \vec{p}[i])^2 * \vec{w}(\vec{p})[i]} \quad (6)$$

where $\vec{w}(\vec{p})$ is the weight vector associated with the prototype $\vec{p}$

For simplicity of the notation, the weight vector of a prototype $\vec{p_i}$, $\vec{w}(\vec{p_i})$ will be also denoted by $\vec{w_i}$. Given the new similarity function, membership function of an instance to a region of a prototype can be defined using the NP rule defined in

$$\begin{aligned} &\forall v = \langle \vec{p_v}, s_v \rangle \quad r_i = \langle \vec{p_i}, s_i, \vec{w_i} \rangle \in C \\ &\quad \text{then} \quad s_v = s_i \quad \text{iff} \\ &\forall r_{i'} = \langle \vec{p_{i'}}, s_{i'}, \vec{w_{i'}} \rangle \in C \quad d_{\vec{w_i}}(\vec{p_v}, \vec{p_i}) \leq d_{\vec{w_{i'}}}(\vec{p_v}, \vec{p_{i'}}). \end{aligned} \quad (7)$$

where $\vec{p_v}$ is the location of the example v and s_v its associated class

This new NP rule shows how the similarity between the instances and each prototype is computed using the weight vector of each prototype. This property provides the method with an interesting advantage: borders generated among the different Voronoi regions are not linear, as is described in Section III-B.

B. Nonlinear Decision Boundary Between Voronoi Regions

Given two prototypes $\vec{p_1}$ and $\vec{p_2}$ with weighting vectors $\vec{w_1}$ and $\vec{w_2}$, respectively, the classification boundary among them is defined by the points $\vec{x}$, which satisfy

$$d_{\vec{w_1}}(\vec{x}, \vec{p_1}) = d_{\vec{w_2}}(\vec{x}, \vec{p_2}). \quad (8)$$

Following the NP rule defined in (5), we can transform the previous equation in

$$\sum_{i=0}^{i<K} (\vec{x}[i] - \vec{p_1}[i])^2 * \vec{w_1}[i] = \sum_{i=0}^{i<K} (\vec{x}[i] - \vec{p_2}[i])^2 * \vec{w_2}[i]. \quad (9)$$

For simplicity, let us assume that $K = 2$. If we develop both sides of the equation, the following is achieved:

$$(\vec{x}[1] - \vec{p_1}[1])^2 * \vec{w_1}[1] + (\vec{x}[2] - \vec{p_1}[2])^2 * \vec{w_1}[2] = (\vec{x}[1] - \vec{p_2}[1])^2 * \vec{w_2}[1] + (\vec{x}[2] - \vec{p_2}[2])^2 * \vec{w_2}[2]. \quad (10)$$

If we put all the terms depending on $\vec{x}[1]$ to one side, and the terms depending on $\vec{x}[2]$ to the other side, we have

$$(\vec{x}[1] - \vec{p_1}[1])^2 * \vec{w_1}[1] - (\vec{x}[1] - \vec{p_2}[1])^2 * \vec{w_2}[1] = (\vec{x}[2] - \vec{p_2}[2])^2 * \vec{w_2}[2] - (\vec{x}[2] - \vec{p_1}[2])^2 * \vec{w_1}[2]. \quad (11)$$

Extending the squared terms, we obtain

$$\begin{aligned}&\vec{x}[1]^2\vec{w_1}[1] - \vec{x}[1]^2\vec{w_2}[1] + \vec{p_1}[1]^2\vec{w_1}[1] - \vec{p_2}[1]^2\vec{w_2}[1] \\ &\quad - 2\vec{x}[1]\vec{p_1}[1]\vec{w_1}[1] + 2\vec{x}[1]\vec{p_2}[1]\vec{w_2}[1] \\ &\qquad = \vec{x}[2]^2\vec{w_2}[2] - \vec{x}[2]^2\vec{w_1}[2] + \vec{p_2}[2]^2\vec{w_2}[2] - \vec{p_1}[2]^2\vec{w_1}[2] \\ &\qquad\quad - 2\vec{x}[2]\vec{p_2}[2]\vec{w_2}[2] + 2\vec{x}[2]\vec{p_1}[2]\vec{w_1}[2]. \end{aligned} \quad (12)$$

Finally, reorganizing the terms, we can obtain the equation of the border, shown in

$$\begin{aligned}&(\vec{w_1}[1] - \vec{w_2}[1])\,\vec{x}[1]^2 + 2\,(\vec{p_2}[1]\vec{w_2}[1] - \vec{p_1}[1]\vec{w_1}[1])\,\vec{x}[1] \\ &\quad + \vec{p_1}[1]^2\vec{w_1}[1] - \vec{p_2}[1]^2\vec{w_2}[1] \\ &\qquad = (\vec{w_2}[2] - \vec{w_1}[2])\,\vec{x}[2]^2 + 2\,(\vec{p_1}[2]\vec{w_1}[2] - \vec{p_2}[2]\vec{w_2}[2])\,\vec{x}[2] \\ &\qquad\quad + \vec{p_2}[2]^2\vec{w_2}[2] - \vec{p_1}[2]^2\vec{w_1}[2]. \end{aligned} \quad (13)$$

We can see that each side of the equation is quadratic and depends on $\vec{x}[1]$ and $\vec{x}[2]$, respectively. Furthermore, we can see how the quadratic terms on both sides depend on the difference between the weights of the prototypes, $(\vec{w_1}[1] - \vec{w_2}[1])$ for the left-hand side of the equation and $(\vec{w_2}[2] - \vec{w_1}[2])$ for the right-hand side. If the weights of both prototypes are the same, for instance, the weight vector $\vec{w}$ as it occurs in global weighting methods, the difference becomes zero, eliminating the quadratic behavior and obtaining (14). This equation is a developed version of the linear boundary equation between Voronoi regions when the Mahalanobis distance is used. If in (14) we make the weight vector to take the value of $(1, 1)$, the Mahalanobis distance reduces to the Euclidean distance [32]

$$\begin{aligned}&2\,(\vec{p_2}[1]\vec{w}[1] - \vec{p_1}[1]\vec{w}[1])\,\vec{x}[1] + \vec{p_1}[1]^2\vec{w}[1] - \vec{p_2}[1]^2\vec{w}[1] \\ &= 2(\vec{p_1}[2]\vec{w}[2] - \vec{p_2}[2]\vec{w}[2])\,\vec{x}[2] + \vec{p_2}[2]^2\vec{w}[2] - \vec{p_1}[2]^2\vec{w}[2]. \end{aligned} \quad (14)$$

Therefore, the distance function can be considered as a generalization of the Mahalanobis distance, where each prototype receives a different weight factor, in a similar way that adaptive metrics for fuzzy clustering introduced in Section II do. Our case is also equivalent to the use of a different covariance matrix for each cluster when generating discriminant functions for a Bayesian classifier, assuming that the class-conditional probability densities are multivariate normal. Therefore, the decision surfaces are hyperquadratics, and may assume any of the general forms: pairs of hyperplanes, hyperspheres, etc., [32].

C. Computing the Weights of the Prototypes

Computing the weighting vectors of the prototypes is based on a local normalization of the data. The normalization is based on the variance of the location of the instances of each region with respect to the average, i.e., the centroid of the region or prototype. From a practical point of view, it is intended to ensure that all the features produce the same contribution to the total distortion. This must be done in a local way, i.e., for each prototype. So, each prototype is given one weighting vector that defines its own similarity metric in order to attract or to move away the training examples.

The average distortion of a prototype (or a region) $r_i = \langle \vec{p_i}, s_i, \vec{w_i} \rangle$, say D_{r_i}, is defined as the average distortion contributed by each training instance $\vec{x}$ belonging to the region with respect to the prototype, as defined in [33]

$$D_{r_i} = \frac{1}{||R_i||} \sum_{\vec{x} \in R_i} d^2_{\vec{w}_i}(\vec{x}, \vec{p_i}) \quad (15)$$

where $||R_i||$ is the size (number of elements) of the set R_i (the set of all the instances for which r_i is the closest prototype).

Furthermore, we can obtain an average distortion vector for this prototype $\vec{D}_{r_i}$, which contains the distortion contributed by each feature, as defined in

$$\vec{D}_{r_i}[j] = \frac{1}{||R_i||} \sum_{\vec{x} \in R_i} (\vec{x}[j] - \vec{p}[j])^2 * \vec{w}[j], \quad \text{for} \quad j = 1, \ldots, K \quad (16)$$

which uses the similarity function defined in (6)

Formally, if we assume that the prototype is the centroid of the region, average distortion is the average squared distance of the location of all the instances in the region from the centroid. Then, the goal of this method is for the values of all the components of the distortion vector of a prototype to be the same. More formally, this can be seen as a normalization of the data in the region of the same variance that mathematically is represented by

$$\vec{D}_{r_i}[0] = \vec{D}_{r_i}[1] = \ldots = \vec{D}_{r_i}[K-1]. \quad (17)$$

The way of computing the weighting vectors derives from (17). For simplicity, suppose $K = 2$ and a prototype $r_i = \langle \vec{p}, s, \vec{w} \rangle$. The previous goal can be defined as

$$\vec{D}_{r_i}[0] = \vec{D}_{r_i}[1] \quad (18)$$

which is transformed using (16) into

$$\sum_{\vec{x} \in R_i} (\vec{x}[0] - \vec{p}[0])^2 * \vec{w}[0] = \sum_{\vec{x} \in R_i} (\vec{x}[1] - \vec{p}[1])^2 * \vec{w}[1]. \quad (19)$$

However, when data is not normalized, the typical situation is that $\vec{D}_r[0] < \vec{D}_r[1]$ or that $\vec{D}_r[0] > \vec{D}_r[1]$. To ensure the equality becomes true, we can add a modification factor $\Delta\vec{w}[1]$, as defined in

$$\sum_{\vec{x} \in R_i} (\vec{x}[0] - \vec{p}[0])^2 * \vec{w}[0] = \sum_{\vec{x} \in R_i} (\vec{x}[1] - \vec{p}[1])^2 * (\vec{w}[1] + \Delta\vec{w}[1]). \quad (20)$$

If we operate $\Delta\vec{w}[1]$ in (20), we obtain

$$\Delta\vec{w}[1] = \frac{\sum_{\vec{x}\in R_i}(\vec{x}[0]-\vec{p}[0])^2 * \vec{w}[0]}{\sum_{\vec{x}\in R_i}(\vec{x}[1]-\vec{p}[1])^2} - \vec{w}[1]. \quad (21)$$

This, generalized for K features, is transformed in (22). The first component ($\vec{x}[0]$) has been taken as a reference (any other component could have been taken)

$$\Delta\vec{w}[j] = \frac{\sum_{\vec{x}\in R_i}(\vec{x}[0]-\vec{p}[0])^2 * \vec{w}[0]}{\sum_{\vec{x}\in R_i}(\vec{x}[j]-\vec{p}[j])^2} - \vec{w}[j]. \quad (22)$$

Furthermore, given that the first feature is taken as reference, we can fix it to a defined value, say 1 ($\vec{w}[0]=1$). Then, (22) can be simplified to

$$\Delta\vec{w}[j] = \frac{\sum_{\vec{x}\in R_i}(\vec{x}[0]-\vec{p}[0])^2}{\sum_{\vec{x}\in R_i}(\vec{x}[j]-\vec{p}[j])^2} - \vec{w}[j]. \quad (23)$$

From (23), we can compute the new weights of any prototype using the formula defined in (24). In the equation, a smoothing parameter $z \in [0,1]$ has been included to adjust the intensity of change of the weights. In the experiments in Section V, we show that by modifying these parameters different results can be obtained

$$\vec{w}[j]^{t+1} = \begin{cases} \vec{w}[j]^t, & \text{when } j=0 \\ \vec{w}[j]^t + z * \Delta\vec{w}[j], & \text{otherwise.} \end{cases} \quad (24)$$

Equation (24) includes a temporal element t. That means that we need to define how often the weights of the prototypes are updated. That depends on how LFW is integrated in an NP algorithm. Section IV defines that for the ENPC algorithm.

IV. LFW IN ENPC

In this section, we introduce the main concepts of the ENPC algorithm. Furthermore, we describe the modifications introduced in the algorithm in order to provide it with the LFW capability.

A. Evolutionary Nearest Prototype Classification

The ENPC is based on the adaptation of a set of prototypes in a competitive domain. The biological inspirations of this algorithm are defined in [34]. The algorithm performs iteratively the evolution of a set of prototypes (initially only one) based on the execution of different prototype operators. The operators allow the prototypes to modify their characteristics in order to improve their quality, where this quality measure is related to their contribution to the classification success of the whole classifier. In this section, we introduce the main concepts of the algorithm, although extensive explanations can be found in [12].

1) Algorithm Basis: The ENPC C is composed of set of N prototypes $C = \{r_1, \ldots, r_N\}$. Each prototype r_i is characterized by its quality, say quality(r_i). This quality is defined as a function of the goodness of the prototype, taking into account the number of patterns in its Voronoi region, and whether those patterns belong to the same class as the prototype or not, as defined in

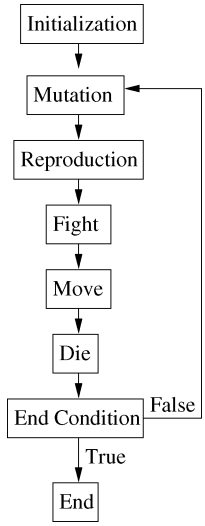

Fig. 1. ENPC algorithm flow.

$$\text{quality}(r_i) = \min\left(1, \text{accuracy}(r_i) * \text{contribution}(r_i)\right). \quad (25)$$

The main idea is that the quality of a prototype is high only if it classifies correctly, what in the equation is called accuracy of the prototype, and if it classifies a sufficient amount of patterns, it is called contribution. The value is limited to the range $[0,1]$. Extended explanation, as well as an exact formulation of how to compute both accuracy and contribution of a prototype, can be found in [12].

Most of the operations that the prototypes can execute depends on this quality measure. The learning process is defined by the flow shown in Fig. 1.

The learning process begins with a classifier of only one prototype. Then, the different operators are executed in the iterative cycle represented in Fig. 1 that ends when a predefined number of iterations has been executed. The goal is that the prototypes evolve until a stable situation is achieved. In that situation, prototypes are supposed to produce a successful classifier. In the evolutionary process, the prototypes are allowed to execute different operators, some of them taken from the literature. This approach uses them from an evolutionary point of view, giving the prototype capabilities to decide when to execute each of the operators and introducing a high level of randomness in those decisions. Next, the different operations that the prototypes can execute are briefly defined. An extended description can be found in [12].

2) Initialization: To avoid the definition of parameters is always a desirable feature of a new algorithm [35], and one of the main features of our method is the absolute lack of initial conditions. These initial conditions, typically summarized in the number of prototypes, the initial set of prototypes, and a smoothing parameter are avoided given the following.

- The initial number of prototypes is always one. The method is able to generate new prototypes stabilizing in

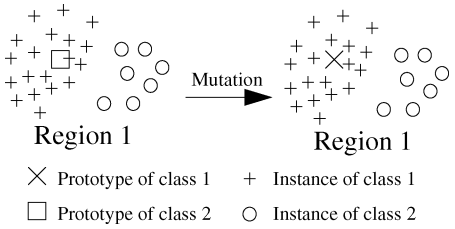

Fig. 2. Example of execution of the mutation operator.

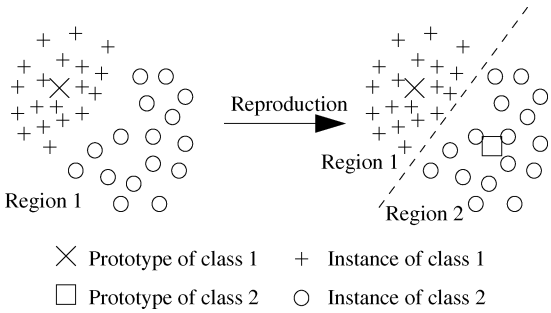

Fig. 3. Example of execution of the reproduction operator.

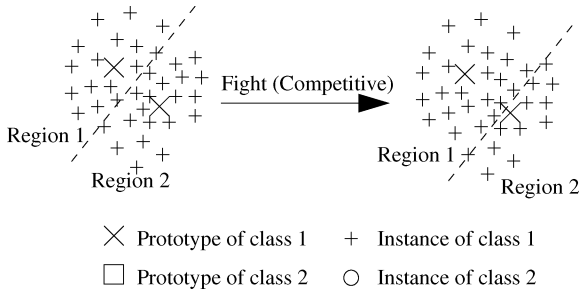

Fig. 4. Example of execution of the fight operator with competition.

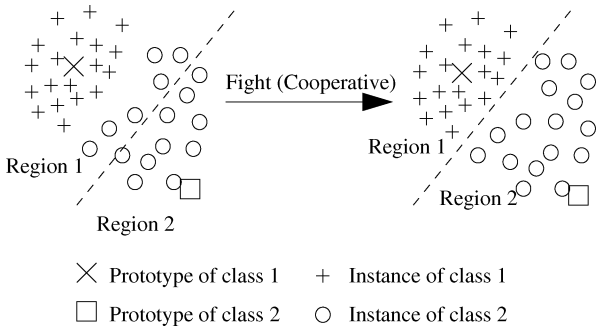

Fig. 5. Example execution of the fight operator with cooperation.

the most appropriate number, in terms of the previously defined "quality" measure.

- The initial location of the only prototype is not relevant (it clusters the whole domain, wherever it is located).
- There are no smoothing parameters. The method automatically adjust the intensity of change in prototypes taking into account their qualities in each iteration.

3) Mutation Operator: The goal of this operator is to label each prototype with the most populate class in each region. Following the NN rule, each prototype knows the number of patterns of each class located in its region. Then, the prototype changes, if needed, and becomes the same class as the most abundant class of patterns in its region. Fig. 2 shows an example of this operator. In the example, a prototype of class 2 changes to class 1, given that it has 19 patterns of class 1 and only 7 of class 2.

This approach to obtain the main class is typically used when unsupervised learning is applied to supervised classification [36], [37].

4) Reproduction Operator: The goal of this operator is to introduce new prototypes in the classifier. The insertion of new prototypes is a decision that is taken by each prototype, in the sense that each prototype has the opportunity of introducing a new prototype in order to increase its own quality. Thus, the regions with patterns belonging to different classes can create new regions containing the patterns of a different class from the class of the prototype, as it is shown in Fig. 3. The probability of reproduction is proportional to the difference in the number of instances belonging to the same and different classes.

5) Fight Operator: This operator provides the prototype with the capability of obtaining patterns from other regions. The steps to execute are defined as follows.

1) Each prototype r_i chooses the prototype $r_{i'}$ against which to fight. Prototypes are chosen from the set of prototypes in its neighborhood. To decide which prototype to choose from the neighbors set, a roulette is used assigning to each region $r_j \in \text{neighbors}(r_i)$ a slice of size proportional to the difference between its quality and the quality of r_i.
2) Decide whether to fight or not. The probability of fighting between prototypes r_i and $r_{i'}$ is proportional to the distance of their qualities.
3) If prototype r_i decides to fight against prototype $r_{i'}$, there are two possibilities. Given s_i as the class associated to r_i and $s_{i'}$ as the class associated to $r_{i'}$, we have the following possibilities.
   a) If $s_i \neq s_{i'}$ (cooperation). Both prototypes belong to different classes. In this case, the prototype $r_{i'}$ will give the prototype r_i the patterns of the class s_i. Fig. 5 shows an execution of this operator, where the prototype 1 that owns patterns of the classes 1 and 2 gives the patterns of class 2 to the prototype 2.
   b) If $s_i = s_{i'}$ (competition). Fig. 4 shows an execution of this operator, where prototype 1 steals some patterns from prototype 2. The winner is decided again by using a roulette with only two slices, each slice belonging to each prototype, and sizes proportional to the qualities of each prototype. Furthermore, the amount of patterns that are transferred depends on a probability proportional to the qualities of both prototypes.

Figs. 4 and 5 show that the translation of patterns from one set to another changes the border between the regions. For example, suppose there were instances of class 2 at the top left of Fig. 5. A cooperative fight would transfer them to the prototype of class 2, even though there is no simple movement of the border that corresponds to this. The real modification of the

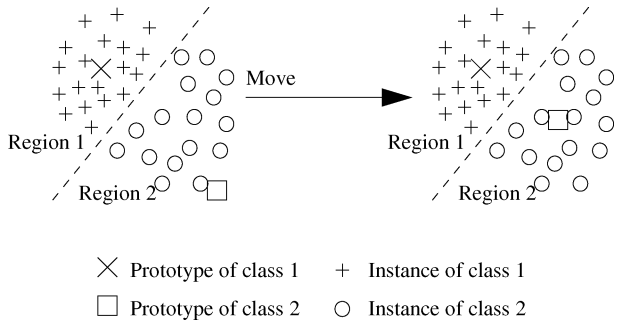

Fig. 6. Example execution of the move operator.

border will be performed with the execution of the move operator, that will move the prototype location, and hence, the borders of the Voronoi regions, as explained in Section IV-A6.

6) Move Operator: The move operation relocates each prototype to the best expected place. This is the centroid of all the training data in its region that belongs to the same class. Fig. 6 shows the execution of the move operator from the situation achieved in Fig. 5. The figure shows how the prototype 2 changes its position to the centroid of the region 2. This operation, based on the second step of Lloyd iteration [38], allows us to make a local optimization of each prototype, increasing the performance of the whole classifier.

7) Die Operator: Probability to die is 1 minus the double of the quality, as defined in (26). Successful prototypes will survive with probability of 1, while useless prototypes with quality less than 0.5 may die

$$P_{\text{die}}(r_i) = \begin{cases} 0, & \text{when quality}(r_i) > 0.5 \\ 1 - 2 * \text{quality}(r_i), & \text{when quality}(r_i) \leq 0.5. \end{cases} \tag{26}$$

8) End Condition and Classifier Selection: In [12], several end conditions as well as methods for selecting the output classifier are defined. However, in that work, it was shown that executing the algorithm a limited number of iterations is a single way to obtain successful results. On the other hand, the algorithm generates a different classifier in each iteration, so any of them can be chosen as an output. However, the best one obtained in training is typically chosen, although sometimes, a validation set can be used to improve generalization capabilities.

B. Applying LFW in NP Classification

A way to apply LFW in ENPC algorithm is by including the computation of the weights of the prototypes (described in Section III) at the end of each iteration, as it is defined in Fig. 7. The weighting vector of the only prototype included at the beginning of the algorithm is set to 1 for all its components.

Weight updates are executed as a new operator of the evolutionary process of the original ENPC algorithm. First, at the end of each iteration, and after executing the die operator, all the prototypes recompute their weight vector following (23) and (24). Second, given that the equation introduces a learning factor z, the original ENPC algorithm can be seen as LFW-NPC with $z = 0$ (no change in the weights is performed). Last, the move operator ensures that prototypes are located in the centroid of the regions if all the instances in the region are of the same class of the prototype. It means that, for such regions, the average distortion defined in (15) agrees with the variance in the location of the instances in each region with respect to the average. Therefore, we can ensure that the normalization process is performed correctly.

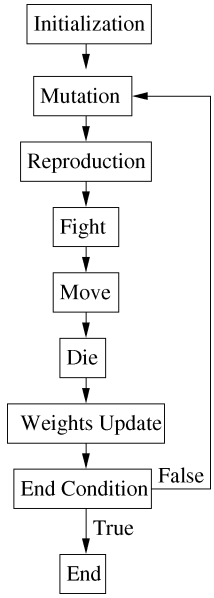

Fig. 7. LFW-NPC algorithm flow.

TABLE I
CLASSIFICATION OF THE LFW-NPC ALGORITHM FOLLOWING DIFFERENT DIMENSIONS

Dimension	Value	Comments
Bias	Preset	Update of feature weights is based on average distortion of the instances in the region of each prototype
Weight Space	Continuous	Weight space is represented with weight vectors composed of continues values
Representation	Given	No transformation of the data is performed
Generality	Local	Each prototype has its own weight vector
Knowledge	Poor	No prior knowledge is introduced

The introduction of this new operator only influences the reproduction operator, in the sense that the new prototype must inherit the weight vector of the ancestor. The other operators remain as before. The evolutionary process ensures that the right weight vectors are computed, obtaining successful results, as will be demonstrated in Section V.

We have shown that the integration of LFW in ENPC is performed very easily. We believe this is an interesting advantage of LFW that could also be integrated with other algorithms very easily. Table I summarizes the characteristics of LFW-NPC following the framework introduced in [10].

V. EXPERIMENTS

In this section, we describe the experiments performed with the LFW-NPC algorithm both in artificial and real data sets. The artificial ones are the XOR data set and waveform, which

TABLE II
EXPERIMENTAL RESULTS ON THE XOR DOMAIN

Domain Characteristics		Naive Bayes	Voted Percep-tron	SMO	IBK (k = 1)	IBK (k = 3)	IBK (k=5)	J48	PART	ENPC	LFW-NPC
[0, 0.5]	DIM2	57.25	50.25	49.15	98.35	98.15	98.35	50.20	50.20	98.55	96.60
	DIM3	46.60	53.95	44.80	96.40	95.90	95.75	50.45	50.45	98.50	98.10
	DIM4	51.15	52.20	47.00	90.80	92.75	93.95	50.35	50.35	98.20	98.55
	DIM5	50.50	53.90	50.80	87.40	89.65	90.75	52.15	52.15	93.75	98.05
[0, 1]	DIM2	62.15	60.05	67.95	98.70	98.40	97.85	51.65	51.65	98.10	97.60
	DIM3	50.10	50.30	43.60	95.85	96.45	96.15	51.05	51.05	96.85	97.55
	DIM4	53.75	51.15	52.65	92.45	93.50	94.60	50.35	50.35	97.15	98.50
	DIM5	52.05	51.75	52.60	90.30	91.40	91.70	50.60	50.60	93.95	98.15
[0, 2]	DIM2	62.15	64.25	67.85	98.70	98.40	97.85	51.65	51.65	98.65	98.05
	DIM3	45.8	51.00	46.45	96.30	96.10	96.05	51.45	51.45	95.70	99.10
	DIM4	53.70	49.80	48.35	92.45	93.50	94.60	50.35	50.35	92.05	98.15
	DIM5	48.30	49.25	46.70	88.20	89.95	90.20	50.10	50.10	85.65	97.20
[0, 10]	DIM2	62.15	59.50	67.90	98.70	98.40	97.85	51.65	51.65	96.45	97.35
	DIM3	66.65	56.90	60.65	95.20	94.80	95.50	62.45	62.15	84.70	97.30
	DIM4	52.05	49.75	52.10	91.35	92.40	93.75	52.15	52.15	64.70	97.70
	DIM5	52.50	52.40	51.50	89.90	91.34	92.80	52.00	52.00	55.45	97.00
[0, 50]	DIM2	38.90	50.40	45.85	98.50	98.20	98.30	51.00	51.00	93.95	97.75
	DIM3	57.35	51.15	68.10	96.50	95.80	96.40	51.50	51.50	54.95	98.75
	DIM4	47.85	49.05	50.35	92.50	93.00	93.25	50.55	50.55	48.35	98.30
	DIM5	50.45	47.50	45.70	89.30	90.00	90.80	51.00	51.00	47.95	98.50

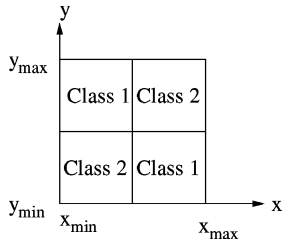

Fig. 8. XOR domain.

have been obtained from the University of California at Irvine (UCI) Machine Learning Repository [39]. The real data sets, also extracted from the UCI, are glass identification, heart disease (Cleveland), Pima Indians diabetes, and liver disorders (Bupa).

The goal of this experimentation is to characterize the types of data sets for which LFW is more powerful than methods based on a nonweighted Euclidean distance. We want to identify in which data sets such algorithms fail, and to verify that LFW improves their result. Specifically, we will compare with an NN approach—the IBK algorithm, and with an NP approach—the ENPC algorithm. The advantage that LFW provides could be extended to other NP-based algorithms in the future. The experiments also demonstrated that the LFW-NPC algorithm works well when compared with other methods. In all the experiments with ENPC and LFW-NPC, we have run both algorithms for 300 iterations.

A. Artificial Data

1) XOR *Domain:* The problem of the ENPC algorithm, similarly to most of the nonweighted NN-rule-based algorithms, appears when not all the features have the same contribution to the classification task. This problem is illustrated with the domain described in Fig. 8, which shows the continuous XOR domain. The data in this data set is continuously and uniformly distributed between predefined values. The data belongs to one of the two different classes, separated by straight line boundaries.

This domain, which seems to be very simple, is a hard domain for several classifiers, mainly when the range of values of the features x and y is different, and when additional irrelevant features are included. For instance, Table II shows the results of executing a tenfold cross validation of several algorithms over a set of 1000 instances randomly generated, following the uniform distribution shown in Fig. 8. In this case, $x_{\min} = 0$ and $x_{\max} = 1$ and $y_{\min}$ and $y_{\max}$ are variables and are defined in the first column of the table. Furthermore, several experiments have been performed adding noisy features. The additional noisy features follow a uniform distribution in the same range defined for feature y. Thus, DIM2 means that the data has only two original dimensions, but DIM3 means that one additional feature has been added, DIM4 means that two features have been added, and so on.

The experimentation has been executed with ENPC and LFW-NPC, using a value of $z = 1$. We also have used other algorithms, like decision trees (J48) [40], decision rules (PART) [41], voted perceptron [42], SVMs [sequential minimal optimization (SMO)] [43], naive Bayes [32], [44], and IBK [45] for different values of k. The implementation of these algorithms is provided by WEKA [45], and they are used with the predefined parameters.

From Table II, several conclusions can be obtained. First, J48, naive Bayes, SMO, and voted perceptron fail in this simple domain, obtaining very poor results with the default parameters.

Second, IBK is not influenced by the range of the y feature given that, by default, the WEKA implementation of IBK normalizes the values of the different features. However, IBK is influenced by the number of dimensions. As the number of dimensions increases, the performance is reduced from around 98% to 89%. Interestingly, as the value of the parameter k becomes lower, the method is more sensitive to the irrelevant features. Last, ENPC is influenced both by the range of the y feature and by the number of noisy features.

TABLE III
EXPERIMENTAL RESULTS OF ENPC/LFW-NPC ALGORITHMS ON THE XOR DOMAIN. IN EACH PAIR OF NUMBERS, THE LEFT NUMBER IS FOR ENPC AND THE RIGHT NUMBER IS FOR LFW-NPC

	Iteration		Learning (%)		Test (%)		Prototypes	
	Average	Average Deviation	Average	Average Deviation	Average	Average Deviation	Average	Average Deviation
DIM2	209.5/144.1	42.30/68.48	97.08/98.12	0.18/0.69	93.95/97.75	0.65/1.35	159.2/8.4	5.64/3.04
DIM3	143.9/138.7	60.26/84.7	91.03/98.47	0.54/0.30	54.95/98.75	1.86/0.65	666.4/5.1	10.32/1.14
DIM4	118.9/134.3	65.86/87.70	91.51/98.62	0.45/0.53	48.35/98.30	2.82/0.98	775.40/4.60	26.20/0.48
DIM5	88.00/87.60	49.20/59.32	90.97/98.32	0.40/0.66	47.95/98.50	4.06/0.7	686.00/5.00	21.00/1.60

Thus, ENPC and IBK fail in this domain, because the nonweighted Euclidean distance fails. The table shows that the results of the LFW-NPC algorithm remain very similar (around 98%) independently of the number or the range of the features that are used. When the dimension of the data set is 2 (no additional features have been added), the results are very close to the results of IBK, but when the number of features increases, the results of the new approach significantly improve the first one. The improvement is higher when compared with its ancestor ENPC, which has worse results when the number of dimensions and the differences among the values of the features increases.

The reason for the strong improvement of LFW-NPC over its ancestor ENPC is illustrated in Table III, which shows the results of executing both the ENPC and LFW-NPC algorithms on the XOR domain, for $x_{\min} = y_{\min} = 0$, $x_{\max} = 1$, and $y_{\max} = 50$. Each row represents the results obtained for the number of dimension indicated, and in each cell, the first value corresponds with ENPC (or LFW-NPC for $z = 0$), and the second with LFW-NPC, for $z = 1$. The table shows the iteration where the best classifier was obtained in training, the success of that classifier on training and test, and the number of prototypes of the classifier. The values shown are the average and standard deviation of ten executions performed in the cross-validation process.

The table shows that the improvement obtained with LFW-NPC over ENPC is very strong, mainly, because of its capability to avoid overfitting, which is illustrated by the following: 1) the difference between test success and training success is around one point in all the cases for LFW-NPC, while for ENPC, it is of more than 40 once an additional feature is inserted and 2) the number of prototypes obtained for LFW-NPC is very small (around 5), and it does not increase with the number of irrelevant features. However, for ENPC, the number of prototypes obtained grows with the number of irrelevant features. Thus, we can say that LFW is required in situations in which the nonweighted Euclidean distance fails, i.e., where the ENPC algorithm fails.

Fig. 9 shows an example of the borders between the Voronoi regions. The four prototypes shown in the figure are obtained by executing the LFW-NPC in the XOR domain with two dimensions (DIM2), $x_{\min} = y_{\min} = 0$, $x_{\max} = 1$ and $y_{\max} = 50$. The figure shows the decision borders between the region generated by prototype $p1$ and the other prototypes, showing that these borders corresponds with parabolas. In this case, the advantage of using weighted Euclidean distance is that the decision boundaries are less influenced by small differences in the location of the prototypes. For instance, prototype 1 is located in the position (0.74, 37.48), while prototype 4 is located in (0.24, 38.81). Although the difference between the y component of the two prototypes seems small (only 1.35), it is huge when compared with the range of the x component, which is only 1. This situation produces a nonweighted Euclidean distance to generate a decision boundary almost horizontal in the figure, producing a very high classification error. Obviously, a method for global weighting should be also useful in this domain, but local weighting is required in many domains.

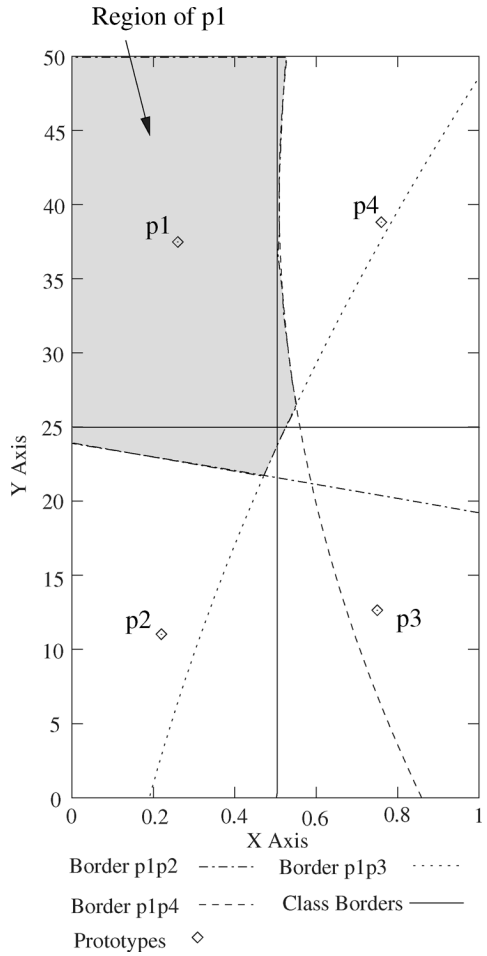

Fig. 9. Nonlinear borders between Voronoi regions in the XOR domain.

2) Waveform: This domain has been obtained from the UCI Machine Learning Repository [39]. The first version of this domain, called waveform-21, consists of 21 relevant features to discriminate three different classes. The 21 features are continuous. The second version adds another 19 irrelevant features. The data set consists of 1000 data and results are obtained from a tenfold cross validation. The goal of this experimentation is to

TABLE IV
COMPARATIVE RESULTS ON THE WAVEFORM-21 AND WAVEFORM-40 DATA SETS

Algorithm	Waveform-21	Waveform-40	Source
Naive Bayes	82.5	79.1	Our (WEKA)
SMO	85.6	84.8	Our (WEKA)
IBK (k=1)	77.1	73.5	Our (WEKA)
IBK (k=3)	78.5	77.8	Our (WEKA)
IBK (k=5)	79.3	78.1	Our (WEKA)
J48	73.1	74.6	Our (WEKA)
PART	75.0	73.0	Our (WEKA)
Relief-F [46]	82.4	83.0	[10]
$K - NN_{VSM}$ [47]	81.6	82.5	[10]
CCF (Cross Category Feature Importance) [48]	76.0	77.9	[10]
VDM (Value Diferenced Metric) [49]	78.4	80.6	[10]
MVDM [50]	78.2	80.9	[10]
MI (Mutual Information) [51], [47]	82.6	82.3	[10]
ENPC (or LFW-NPC with $z = 0$)	82.0	83.4	Our
LFW-NPC ($z = 1$)	83.5	83.6	Our

compare the results obtained by LFW-NPC with the results obtained by different methods of features selection and weighting included in [10]. Table IV summarizes the results.

From the results, several following conclusions can be obtained.

1) SMO obtains the best results, both for waveform-21 and waveform-40.
2) Naive Bayes is influenced by the 19 more irrelevant features included in waveform-40, decreasing its performance in around three points.
3) IBK obtains results under 80% in all the cases and, in the same way as in the XOR domain, the influence of the irrelevant features is stronger for lower values of k.
4) Decision trees and rules (J48 and PART, respectively) also obtain poor results, but they do not seem to be very influenced by the irrelevant features.
5) Both relief-F and k-NN$_{\mathrm{VSM}}$ obtain good results both for waveform-21 and waveform-40. The common element of both algorithms is that they are performance bias methods, as described in [10].
6) In this case, preset bias methods [CCF, VDM, modified value differenced metric (MVDM), and MI] also obtain similar results for both data sets, showing that their capability to manage irrelevant features is also high.
7) ENPC obtains very good results both in waveform-21 and waveform-40.
8) LFW-NPC (with $z = 1$) obtains the second best result for both waveform-21 and waveform-40, showing that it is not influenced by the irrelevant features. This algorithm is also a preset bias algorithm, given that it does not follow a performance-based policy to weight the features, but the weighting process is only guided by the average distortion, as it was defined in Table I.

Opposite to the XOR domain, in this domain, the nonweighted Euclidean distance seems to work well, so LFW does not provide a strong improvement. Section V-B describes additional experimentation to show the performance of LFW over real data sets.

B. Real Data Sets

This section describes the results obtained by the LFW-NPC algorithm in several data sets. The data sets are composed of real data, and have been extracted from the UCI Machine Learning Repository. The domains are Pima Indians diabetes, glass identification, heart disease (Cleveland), and liver disorders (Bupa). The characteristics of each domain (number of instances, attributes, and classes) are summarized in Table V.

TABLE V
SUMMARY OF THE DATA SETS

Domain	# Instances	# attributes	# classes
Glass	214	10	7
Bupa	345	6	2
Cleveland	297	13	2
Pima Indians	768	8	2

First, we are interested in evaluating if the LFW-NPC algorithm is very sensitive to its parameter z, and we use the Pima Indian diabetes data set for such purpose. Table VI shows the results for different z values in the Pima Indian diabetes data set when a tenfold cross validation is performed and repeated ten times (due to the stochastic behavior of the ENPC algorithm). For each value of z, the table shows the execution time, in seconds, of the LFW-NPC algorithm (on an Intel Pentium 4, 3.6 GHz), the performance on learning, on test, and the number of prototypes of the classifier obtained. For all of them, the average and the standard deviation is provided.

The results show several important conclusions. First, higher values of the z parameter produce higher test accuracy, which grows from 67.55% for $z = 0$ up to the 74.39% obtained for $z = 0.7$. Another interesting element is the number of prototypes. Notice that for $z = 1$, the number of prototypes obtained by the classifier is only 5.31, while with $z = 0$ it generates around 140. This reduction in the number of prototypes is a signal of the generalization capabilities of the algorithm, which are also reflected in the differences among the results on training and test. For instance, for $z = 0$, difference between test and trial success is around 23 points, while for $z = 1$, the difference is lower than 4. That shows that LFW-NPC is able to avoid overfitting to the training data, obtaining very successful results on test.

Another interesting result is that, when performing a t-test with a 95% confidence level, the accuracy in test obtained when $z = 1$ is only significantly different than the one obtained when $z = 0$ and $z = 0.1$. Thus, we can say that the accuracy obtained is not very sensitive to the z parameter.

Last, execution time depends on the number of prototypes obtained. If such number is low, as is the case of $z = 0.3$ and

TABLE VI
EXPERIMENTAL RESULTS ON THE PIMA INDIANS DIABETES DATA SET

	Learning Time (seconds)		Learning (%)		Test (%)		Prototypes	
z	Average	Average Deviation	Average	Average Deviation	Average	Average Deviation	Average	Average Deviation
0	7.61	0.94	90.11	1.19	67.55	5.69	142.19	14.69
0.1	2.96	0.89	84.84	1.68	71.59	4.48	64.50	18.51
0.3	0.78	0.13	78.68	0.78	73.50	5.06	13.79	5.78
0.5	0.66	0.08	77.92	0.78	73.70	4.94	9.25	3.92
0.7	0.58	0.07	77.56	0.77	74.39	5.53	7.28	2.51
0.9	0.54	0.06	77.44	0.75	74.20	5.39	6.18	1.93
1	0.55	0.05	77.30	0.73	74.11	5.16	5.31	1.92

TABLE VII
COMPARATIVE RESULTS OVER PIMA INDIAN DIABETES DATA SET

Algorithm	**Success**	**Source**
Naive Bayes	75.69	Our (WEKA)
Voted Perceptron	65.49	Our (WEKA)
SMO	76.63	Our (WEKA)
IBK (k=1)	70.44	Our (WEKA)
IBK (k=3)	73.88	Our (WEKA)
IBK (k=5)	73.43	Our (WEKA)
J48	74.48	Our (WEKA)
PART	74.18	Our (WEKA)
CNN (Condensed Nearest Neighbor Rule) [52]	69,78	[53]
DEL (Decremental Encoding Length) [53]	71.61	[53]
DROP 5 (Decremental Reduction Optimization) [53]	73.05	[53]
SNN (Selective Nearest Neighbor Rule) [54]	67.97	[53]
ESOM (Evolutionary Self-Organizing Maps) [55]	78.4	[55]
LVQ (Learning Vector Quantization) [56]	75.8	[55]
ENPC (or LFW-NPC with $z = 0$)	67.55	Our
LFW-NPC ($z = 1$)	74.12	Our

TABLE VIII
AVERAGE CLASSIFICATION ACCURACY OF SEVERAL ALGORITHMS ON DIFFERENT DATA SETS

Domain	LFW-NPC	ENPC	J48	PART	Naive Bayes	IBK ($k = 1$)	IBK ($K = 3$)	SMO
Glass	63.714	70.571^{--}	68.359^{--}	70.810^{--}	48.491^{++}	69.173^{--}	69.630^{--}	57.481^{++}
Bupa	65.294	$63.735^{=}$	$66.008^{=}$	63.084^{+}	55.625^{++}	62.895^{+}	$63.161^{=}$	57.954^{++}
Cleveland	79.586	60.103^{++}	77.107^{++}	$78.143^{=}$	83.384^{--}	75.823^{++}	$80.494^{=}$	83.829^{--}
Pima Indians	74.118	67.553^{++}	74.482^{-}	$74.182^{=}$	$75.693^{=}$	70.444^{++}	73.881^{+}	76.631^{--}
Waveform-21	83.5	$82.0^{=}$	73.1^{++}	75.00^{++}	82.5^{++}	77.1^{++}	78.5^{++}	85.6^{-}
Waveform-40	83.6	$83.4^{=}$	74.6^{++}	73.0^{++}	79.1^{++}	73.5^{++}	77.8^{++}	84.8^{-}
XOR [0,50] DIM5	98.5	47.95^{++}	51.05^{++}	51.00^{++}	50.45^{++}	89.30^{++}	90.00^{++}	45.7^{++}

higher, execution time is under 1 s. While the number of prototypes obtained grows, the execution time also grows. Thus, although LFW-NPC adds some complexity to the ENPC algorithm, due to the reduction on the number of prototypes of the evolving classifier, the resulting execution time is much smaller.

We have compared these results with other classification algorithms, summarizing the results in Table VII. We can see that there are other algorithms that improve the results of LFW-NPC, but LFW-NPC obtains the result close to the best one.

Given that small variations in the z parameter do not provide significant differences in the accuracy, for the rest of domains, we have executed the LFW-NPC algorithm with the value of $z = 1$. For comparison reasons, we also include the result when $z = 0$ (equivalent to ENPC). Furthermore, we have executed J48, PART, naive Bayes, SMO, and IBK for $k = 1$ and $k = 3$, all of them with the default parameters defined in WEKA. We execute the default parameter setting given we do not perform any parameter optimization in the LFW-NPC algorithm either. The only parameter set for LFW-NPC and ENPC is the number of iterations, which has been set to 300 in all the cases. For all the algorithms and domains, a tenfold cross validation has been performed. In addition, given that some of the algorithms have a stochastic behavior, we repeat the learning and test processes of each fold ten times. Table VIII summarizes the results obtained for each algorithm. The reported value is the average of the 100 executions.[2] Table IX reports the standard deviation of each series of the new experiments.

A two-tailed t-test has been performed to study whether the differences of the results are significant or not. Specifically, we are interested in knowing when the results of the LFW-NPC algorithm are significantly better or worse than the results of the other algorithms for each domain. Thus, following the notation introduced in [53], the superscripts "$+$" and "$++$" indicate that LFW-NPC's average accuracy was significantly higher that the other method's average accuracy at a 90% and 95% confidence level, respectively. Similarly, superscripts "$-$" and "$--$" indicate that LFW-NPC's average accuracy was significantly lower

[2]The use of the same training and testing set in each group of ten repetitions could overestimate the significance of differences due to stochasticity of the method, underestimating the variations in training and test data distributions.

TABLE IX
STANDARD DEVIATION OF THE CLASSIFICATION ACCURACY OF SEVERAL ALGORITHMS ON DIFFERENT DATA SETS

Domain	LFW-NPC	ENPC	J48	PART	Naive Bayes	IBK ($k = 1$)	IBK ($K = 3$)	SMO
Glass	8.871	10.816	9.638	10.087	9.152	9.077	8.978	9.279
Bupa	9.884	6.327	8.389	8.022	9.178	7.812	8.178	0.956
Cleveland	6.174	7.982	7.503	6.695	5.775	7.190	7.251	5.706
Pima Indians	5.108	5.692	5.071	4.340	4.405	4.699	4.256	4.016

that the other method's average accuracy at a 90% and 95% confidence level. The superscript "=" indicates that differences are not significant at such confidence levels.

From these results, we can evaluate when our method for LFW should be applied, and when it should not. LFW provides additional capabilities to NP methods based on a nonweighted Euclidean distance, so LFW should be useful in domains where such a distance metric fails. A way to evaluate when such a distance fails is by studying when the methods that use it fail, i.e., when NP (ENPC) and NN (IBK) algorithms fail. When we have a new and unknown domain, we cannot use test data to do this evaluation. However, we can always use a validation set obtained from training data, in a similar way as it was used in [12]. The obtained results in the validation can be used as a way of telling when to use LFW-NPC without using test data results.

For instance, in the glass data set, both ENPC and IBK obtain very good results, around 70%, which is a value similar to the ones obtained by J48 and PART, and higher than the ones obtained by SMO and naive Bayes. In this domain, LFW-NPC obtains a poor result, similar to SMO and naive Bayes. In the BUPA data set, ENPC and IBK also obtain good results and, in this case, LFW-NPC also does. However, in the Cleveland domain, ENPC and IBK with $k = 1$ fail, obtaining the worst results. That means that the distance metric is failing. In this case, LFW-NPC improves the result of ENPC in almost 20 points. Last, in the Pima Indians data set, where ENPC and IBK also obtain the worst results, LFW-NPC significantly improves them.

These results extend the conclusions obtained in the XOR domain, where we demonstrated that LFW-NPC obtained very good results in the data sets where methods based on the nonweighted Euclidean distance failed. The failure of the distance metric can be identified both through the results of IBK and ENPC. In the case of IBK, a fail in the distance metric produces a very low classification accuracy for $K = 1$. Such result may be improved with a higher value of K. In the case of ENPC, a fail in the distance metric produces both a huge number of prototypes and very high differences between the accuracy on training and the accuracy on test. In such cases, LFW provides better results.

VI. CONCLUSION

In many real classification problems, it is usual to find domains that contain a great number of attributes. These attributes sometimes are essential in the classification task, but at other times are superfluous. In addition, it could be the case that a dependency, more or less complex, exists between the attributes, in such a way that it makes difficult both the classification task and the detection of the irrelevant attributes. It could also be possible that the ranges of values of the attributes are very different.

NN and NP approaches use a distance function in the n-dimensional space defined by the features of the data. This function is affected by all the features of the data set that, as defined previously, may include noisy and/or irrelevant information which could make the distance function fail. As an example, in a bidimensional space in which one of the dimensions has a rank much greater than the other, the Euclidean distance will detect differences in the greatest dimension, although the classification can depend on both. This is the reason why the systems based on neighborhoods give poor results when they are evaluated in domains with these characteristics. Furthermore, these differences are not uniform in the whole space, but they depend on small regions in such a space, and therefore, global weighting methods cannot be successfully applied.

We have described a new method for LFW in NP classification. The main ideas of the method are as follows: 1) each prototype owns a weight vector, so the NN rule is based on a locally weighted similarity function and 2) weight vector computation is performed based on local normalization of the data. The method generates Voronoi regions whose borders are not linear. This property is very important, because it allows the method to generate nonlinear partitions of the space that adapt better to the real data generated by the problem.

We have integrated our method for LFW in a previous algorithm, called ENPC, which has proved to obtain good results following an iterative process where a set of prototypes evolves by executing several operators to modify their characteristics, like location, class, etc., [12]. A new phase, called prototype weights update, is added in each iteration to modify the weight vector of each prototype, obtaining the new version of the algorithm, called LFW-NPC.

Experimental results have shown that the methods that use the nonweighted Euclidean distance (as ENPC and IBK) fail in the same domains due to the distance metric. In such domains, LFW-NPC improves the results of its ancestor. The improvements are based on a higher generalization capability, reflected in its capability to generate classifiers with less prototypes, and to generate a lower difference among training and test results.

Analyzing the results of Table VIII, it is possible to remark that there is not a method that performs better than the rest in all domains. In a first look, it could be concluded that SMO performs better in most of the domains. However, experiments prove that in such domains in which SMO performs not well, it performs worse, showing a very high domain dependence. By the opposite, our approach has a more regular behavior in all domains. It can be observed how the performance of LFW-NPC is either optimal or near the optimum in the cases where some other technique performs better.

The results yield the conclusion that when there is no domain knowledge that can tell what is the best method, our approach can be the best choice, because of its characteristic of regularity in the performance.

ACKNOWLEDGMENT

The authors would like to thank N. Crespo for her help in the development of the LFW-NPC algorithm.

REFERENCES

[1] J. C. Bezdek and L. I. Kuncheva, "Nearest neighbour classifier designs: An experimental study," *Int. J. Intell. Syst.*, vol. 16, pp. 1445–1473, 2001.

[2] L. I. Kuncheva and J. C. Bezdek, "Nearest prototype classification: Clustering, genetic algorithms, or random search?," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 28, no. 1, pp. 160–164, Feb. 1998.

[3] T. Kohonen, *Self-Organization and Associative Memory*, 3rd ed. Berlin, Germany: Springer-Verlag, 1984, 1989.

[4] S. Seo and K. Obermayer, "Soft learning vector quantization," *Neural Comput.*, vol. 15, no. 7, pp. 1589–1604, 2003.

[5] R. Duda, P. Hart, and D. Stork, *Pattern Classification*, 2nd ed. New York: Wiley, 2001.

[6] R. Bellman, *Adaptive Control Process*. Princeton, NJ: Princeton Univ. Press, 1961.

[7] J. Li, M. T. Manry, P. L. Narasimha, and C. Yu, "Feature selection using a piecewise linear network," *IEEE Trans. Neural Netw.*, vol. 17, no. 5, pp. 1101–1115, Sep. 2006.

[8] K. Fukumizu, F. Bach, and M. Jordan, "Dimensionality reduction for supervised learning with reproducing kernel Hilbert spaces," *J. Mach. Learn. Res.*, vol. 5, pp. 73–99, 2004.

[9] K. Torkkola, "Feature extraction by non-parametric mutual information maximization," *J. Mach. Learn. Res.*, vol. 3, pp. 1415–1438, 2003.

[10] D. Wettschereck, D. Aha, and T. Mohri, "A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms," *Artif. Intell. Rev.*, vol. 11, pp. 273–314, 1997.

[11] D. Aha and R. Goldstone, "Concept learning and flexible learning," in *Proc. 14th Annu. Conf. Cogn. Sci. Soc.*, 1992, pp. 534–539.

[12] F. Fernández and P. Isasi, "Evolutionary design of nearest prototype classifiers," *J. Heuristics*, vol. 10, no. 4, pp. 431–454, 2004.

[13] F. Bach and M. Jordan, "Kernel independent component analysis," *J. Mach. Learn. Res.*, vol. 3, pp. 1–48, 2002.

[14] C. Estébanez, J. Valls, R. Aler, and I. Galván, "A first attempt at constructing genetic programming expressions for EEG classification," in *Lecture Notes in Computer Science*. Berlin, Germany: Springer-Verlag, 2005.

[15] F. Ricci and P. Avesani, "Data compression and local metrics for nearest neighbor classifiers," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 21, no. 4, pp. 380–384, Apr. 1999.

[16] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning," *Artif. Intell. Rev.*, vol. 11, pp. 11–73, 1997.

[17] J. Peng, D. Heisterkamp, and K. Dai, "LDA/SVM driven nearest neighbor classification," *IEEE Trans. Neural Netw.*, vol. 14, no. 4, pp. 940–942, Jul. 2003.

[18] T. Hastie and R. Tibshirani, "Discriminant adaptive nearest neighbor classification," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 18, no. 6, pp. 607–616, Jun. 1996.

[19] C. Domeniconi and D. Gunopulos, "Adaptive nearest neighbor classification using support vector machines," in *Advances in Neural Information Processing Systems*. Cambridge, MA: MIT Press, 2001, vol. 14.

[20] C. Domeniconi, J. Peng, and D. Gunopulos, "An adaptive metric for pattern classification," in *Advances in Neural Information Processing Systems*. Cambridge, MA: MIT Press, 2000, vol. 13.

[21] C. Domeniconi and D. Gunopulos, "Efficient local flexible nearest neighbor classification," in *Proc. 2nd SIAM Int. Conf. Data Mining (SDM)*, 2002.

[22] C. Domeniconi, J. Peng, and D. Gunopulos, "Locally adaptive metric nearest neighbor classification," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 9, pp. 1281–1285, Sep. 2002.

[23] C. Domeniconi, D. Gunopulos, and J. Peng, "Large margin nearest neighbor classifiers," *IEEE Trans. Neural Netw.*, vol. 16, no. 4, pp. 899–909, Jul. 2005.

[24] T. Liu, A. W. Moore, A. G. Gray, and K. Yang, "An investigation of practical approximate nearest neighbor algorithms," in *Proc. Neural Inf. Process. Syst. (NIPS)*, 2004, pp. 825–832.

[25] M. Pregenzer, G. Pfurtscheller, and D. Flotzinger, "Automated feature selection with distinction sensitive vector learning quantization," *Neurocomputing*, vol. 11, pp. 19–29, 1996.

[26] B. Hammer and T. Villmann, "Generalized relevance learning vector quantization," *Neural Netw.*, vol. 15, pp. 1059–1068, 2002.

[27] A. S. Sato and K. Yamada, "Generalized learning vector quantization," in *Advances in Neural Information Processing Systems*, G. Tesauro, D. Touretzky, and T. L. , Eds. Cambridge, MA: MIT Press, 1995, vol. 7, pp. 423–429.

[28] T. Bojer, B. Hammer, D. Schunk, and K. Tluk, "Relevance determination in learning vector quantization," in *Proc. Eur. Symp. Artif. Neural Netw. (ESANN)*, Brussels, Belgium, 2001, pp. 271–276.

[29] F. Scheleif, T. Villmann, and B. Hammer, "Local metric adaptation for soft nearest prototype classification to classify proteomic data," in *Lecture Notes on Artificial Intelligence*, ser. 3849. Berlin, Germany: Springer-Verlag, 2006, pp. 290–296.

[30] D. E. Gustafson and W. Kessel, "Fuzzy clustering with a fuzzy covariance matrix," in *Proc. IEEE Conf. Decision Control*, 1979, pp. 761–766.

[31] I. Gath and A. Geva, "Unsupervised optimal fuzzy clustering," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 11, no. 7, pp. 773–781, Jul. 1989.

[32] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. New York: Wiley, 1973.

[33] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Norwell, MA: Kluwer, 1992.

[34] F. Fernández and P. Isasi, "Automatic finding of good classifiers following a biologically inspired metaphor," *Comput. Inf.*, vol. 21, no. 3, pp. 205–220, 2002.

[35] E. Berglund and J. Sitte, "The parameterless self-organizing map algorithm," *IEEE Trans. Neural Netw.*, vol. 17, no. 2, pp. 305–316, Mar. 2006.

[36] S. Bermejo and J. Cabestany, "A batch learning algorithm vector quantization algorithm for nearest neighbour classification," *Neural Process. Lett.*, vol. 11, pp. 173–184, 2000.

[37] N. R. Pal, J. C. Bezdek, and E. C.-K. Tsao, "Generalized clustering networks and Kohonen's self-organizing scheme," *IEEE Trans. Neural Netw.*, vol. 4, no. 4, p. 1993, Jul. 1993.

[38] S. P. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inf. Theory*, no. IT-28, pp. 127–135, Mar. 1982.

[39] C. L. Blake and C. J. Merz, "UCI Repository of Machine Learning Databases," 1998 [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html

[40] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann, 1993.

[41] E. Frank and I. H. Witten, "Generating accurate rule sets without global optimization," in *Proc. 15th Int. Conf. Mach. Learn.*, 1998, pp. 144–151.

[42] Y. Freund and R. E. Schapire, "Large margin classification using the perceptron algorithm," *Mach. Learn.*, vol. 3, no. 37, pp. 277–296, 1999.

[43] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines*. Cambridge, U.K.: Cambridge Univ. Press, 2000.

[44] G. H. John and P. Langley, "Estimating continuous distributions in Bayesian classifiers," in *Proc. 11th Conf. Uncertainty Artif. Intell.*, 1995, pp. 338–345.

[45] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*. New York: Elsevier, 2005.

[46] K. Kira and L. A. Rendell, "A practical approach to feature selection," in *Proc. 9th Int. Conf. Mach. Learn.*, 1992, pp. 249–256.

[47] D. Wettschereck, "A description of the mutual information approach and the variable similarity metric," German Nat. Res. Center Comput. Sci., Artif. Intell. Res. Div., Sankt Augustin, Germany, Tech. Rep. 944, 1995.

[48] R. H. Creecy, B. M. Masand, S. J. Smith, and D. L. Waltz, "Trading mips and memory for knowledge engineering," *Commun. ACM*, vol. 35, 1992.

[49] C. Stanfill and D. Waltz, "Toward memory-based reasoning," *Commun. ACM*, vol. 29, pp. 1213–1228, 1986.

[50] S. Cost and S. Salzberg, "A weighted nearest neighbor algorithm for learning with symbolic features," *Mach. Learn.*, vol. 10, pp. 57–78, 1993.

[51] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Technol. J.*, vol. 27, pp. 379–423, 1948.

[52] P. E. Hart, "The condensed nearest neighbor rule," *IEEE Trans. Inf. Theory*, vol. IT-14, no. 3, pp. 515–516, May 1968.

[53] D. R. Wilson and T. R. Martinez, "Reduction techniques for instance based learning algorithms," *Mach. Learn.*, vol. 38, pp. 257–286, 2000.

[54] G. L. Ritter, H. B. Woodruff, S. R. Lowri, and T. L. Isenhour, "An algorithm for a selective nearest neighbor decision rule," *IEEE Trans. Inf. Theory*, vol. IT-21, no. 6, pp. 665–669, Nov. 1975.

[55] D. Deng and N. Kasabov, "On-line pattern analysis by evolving organizing maps," *Neurocomputing*, vol. 51, pp. 87–103, 2003.

[56] T. Kohonen, *Self-Organizing Maps*. Berlin, Germany: Springer-Verlag, 1995.

Fernando Fernández received the B.S. and Ph.D. degrees in computer science from Universidad Carlos III de Madrid (UC3M), Madrid, Spain, in 1999 and 2003, respectively. In his Ph.D. dissertation, he studied different nearest prototype approaches for the discretization of the state space in reinforcement learning problems.

He has been a member of the faculty of the Computer Science Department, UC3M, since October 2005. In 2001, he became an Assistant and Associate Professor. In fall 2000, he was a Visiting Student at the Center for Engineering Science Advanced Research, Oak Ridge National Laboratory, Oak Ridge, TN. He was also a Postdoctoral Fellow at the Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, from October 2004 to December 2005. He published more than 30 journal and conference papers, mainly in the field of machine learning and planning. His interests are in intelligent systems that operate in continuous and stochastic domains. His research is also focused on nearest prototype approaches for data analysis, both with classical attribute value and relational representations.

Dr. Fernández is a recipient of a predoctoral FPU fellowship award from Spanish Ministry of Education (MEC), a Doctoral Prize from UC3M, and a MEC-Fulbright postdoctoral Fellowship.

Pedro Isasi received the computer science degree and the Ph.D. degree from the Universidad Politécnica de Madrid (UPM), Madrid, Spain in 1994.

Currently, he is a Full Professor at the Computer Science Department, Universidad Carlos III de Madrid (UC3M), Madrid, Spain, where he is the Head of the Neural Networks and Evolutionary Computation Laboratory (ENANNAI). His main research areas are the artificial intelligence, the optimization problems, automatic learning, all mixed with evolutionary computation techniques and neural networks.

Dr. Isasi is the Chief of the Computational Finance and Economics Technical Committee (CFETC) of the IEEE Computational Intelligence Society (CIS).